Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA - RELAZIONE DI TIROCINIO

Progettazione ed implementazione di un modulo di interfaccia grafica web 2.0 per BROADgen®, un framework Java™ orientato allo sviluppo di applicazioni web enterprise

Relatore: Prof. Sergio Congiu

Laureando: Paolo Albieri

ANNO ACCADEMICO 2011/2012

A mia nonna

Indice

Introduzione

L'attività di tirocinio si è svolta nell'ambito della programmazione Java e sviluppo web su un arco di tempo lungo 4 mesi, da aprile 2012 a inizio agosto 2012, per un ammontare complessivo di circa 500 ore, presso la Broadway Solutions S.r.l. Essa è un'azienda di piccole dimensioni con sede a Rovigo, tra le prime in Italia ad adottare il modello architetturale dell'Application Service Providing (ASP) per erogare i propri servizi, e si occupa prevalentemente dello sviluppo, della configurazione e della gestione di software gestionale per diversi ambiti aziendali.

Per lo sviluppo delle applicazioni web viene utilizzato un framework proprietario, il "BROADgen®" che genera automaticamente il codice a partire dall'indicazione delle regole di business.
L'obiettivo dell'attività di tirocinio è stato quello di progettare un modulo di interfaccia grafica per BROADgen® basato sulle caratteristiche del web dinamico 2.0 che vada a sostituirsi alla precedente gestione dell'interfaccia, che poggia sul web 1.0.
Il suddetto modulo è stato denominato "BROADGWT®" in relazione al fatto che per svilupparlo si è utilizzato il set di strumenti messi a disposizione da Google™ per lo sviluppo di applicazioni web: Google Web Toolkit. La scelta strategica di utilizzare questo tipo di strumento di sviluppo è stata presa dall'azienda stessa in quanto già utilizzata per altre applicazioni e considerata dotata di un notevole potenziale innovativo.

Durante la prima fase del tirocinio è avvenuto un periodo di formazione da parte del tutor e di altro personale tecnico dell'azienda in cui si sono introdotte al laureando le tecnologie che sarebbero state utilizzate durante il periodo di stage e sono state fornite alcune basi teoriche riguardo a pattern di programmazione, utilizzo dei database e meccanismi di richiesta/risposta. Questo per poter comprendere al meglio il funzionamento di alcune web applications prodotte che sono state poi analizzate ponendo in rilievo il ruolo dei singoli componenti architetturali.
Alla fase di introduzione teorica è seguita la fase di sviluppo vera a propria, che ha occupato l'intera parte rimanente delle ore previste ed è stata realizzata, per

certi versi autonomamente dal laureando e per altri aspetti in collaborazione con il resto del personale tecnico.

Nel primo capitolo verrà esaminata la situazione di partenza delle applicazioni dell'azienda: come sono strutturate e il loro funzionamento. Verrà inoltre specificato quale sarà l'ambito di intervento nel quale si è inserita l'attività di sviluppo da parte del laureando.
In seguito verrà esposta una panoramica sulle varie tecnologie utilizzate durante lo stage e il ruolo che esse svolgono nel contesto dell'applicazione con un riferimento a dove reperire la documentazione relativa.
Infine sarà illustrato il lavoro di sviluppo prendendo qualche esempio pratico di implementazione a livello di codice Java.
Per finire nelle conclusioni saranno riassunti gli obiettivi raggiunti ed il grado di successo del lavoro svolto.

Capitolo 1 – La situazione di partenza

1.1 - L'architettura delle applicazioni Broadway Solutions Srl

Le applicazioni prodotte dalla Broadway Solutions Srl si basano sul tipico modello dell'architettura a 3 livelli (3-tiers), in cui:

- First tier: è costitiuito da un "thin client" che può essere il browser web come anche un dispositivo mobile.
- Second tier: è rappresentato da Apache Tomcat, l'application server su cui vengono eseguite le applicazioni.
- Third tier: è il database server, che utilizza come RDBMS Oracle oppure PostgreSQL, a seconda dei casi.

Più specificatamente, il modello architetturale vero e proprio è il cosiddetto MVC (Model View Controller). L'utilità di questo modello è facilmente comprensibile considerando il classico comportamento di una applicazione web: un client, tipicamente un browser, inoltra la richiesta ad un server per una pagina HTML. Il server ospita un'applicazione scritta in un linguaggio di programmazione lato server il quale preleva i dati da un database, li elabora e li restituisce al client in formato HTML.

La parte più "attiva" in questo procedimento è l'applicazione Web che ha il compito di reperire ed inviare le informazioni. Se venisse utilizzata una singola pagina Web che svolga tutti i compiti descritti in precedenza creando un unico blocco di codice (la pagina), tale codice risulterebbe molto complesso e difficilmente comprensibile oltre a complicare la manutenzione dell'applicazione stessa. [1]

Il modello MVC, e più in generale l'architettura a tre livelli, permette la separazione netta tra i componenti software che gestiscono il modo di presentare i dati, e i componenti che gestiscono i dati stessi.

Nello specifico:

- **Model**: definisce i dati e le operazioni che possono essere eseguite su questi. Quindi definisce le regole di business per l'interazione con i dati, esponendo al View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento. Il Model può inoltre avere la responsabilità di notificare ai componenti del View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere al View di presentare agli occhi degli utenti dati sempre aggiornati.

- **View**: La logica di presentazione dei dati viene gestita solo e solamente dal View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (GUI) che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema. Ogni GUI può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione. [2]

- **Controller**: riceve i comandi dell'utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.

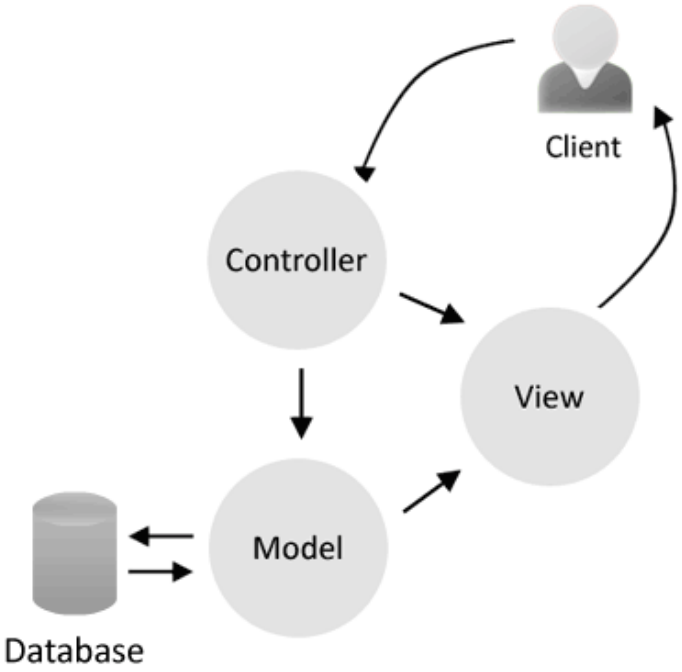

Figura 1.1 - **Divisione dei ruoli nel pattern MVC**

Nello sviluppare il codice secondo questo schema, ci si può concentrare su un problema specifico ed avere la sicurezza che l'intervento rimanga circoscritto al componente di cui ci si sta occupando, lasciando intatti gli altri. In progetti di grandi dimensioni, in cui ogni parte viene creata e mantenuta da persone eventualmente diverse, la divisione logica del codice in zone distinte aumenta l'efficienza complessiva dell'applicazione.

Riferendoci in particolare al contesto della Broadway Solutions Srl, al pervenire di una richiesta al Controller, esso istanzia un oggetto della classe "Command" attraverso l'utilizzo del pattern di programmazione denominato "riflessione" (reflection) e ne

richiama il metodo *execute()*. Tale pattern permette di esaminare, in fase di esecuzione, le classi presenti per poi istanziare dinamicamente un oggetto della classe d'interesse.

Per quanto riguarda il Model, le classi presenti in esso corrispondono alle tabelle residenti nel database (chiamate anche "BusinessObject"), e il framework BROADgen®, in particolare, genera due istanze chiamate "*NomeTabellaDOP.java"* e "*NomeTabellaDO.java"* per ogni singolo record di una generica tabella "*NomeTabella"*. Inoltre viene generata un'unica istanza "*NomeTabellaDOF.java"* per la tabella stessa *NomeTabella*, utlizzando il pattern di programmazione denominato "Singleton", che prevede la creazione di un'unica istanza per una determinata classe od oggetto e il quale si assicura che qualsiasi tentativo di creazione faccia riferimento alla medesima istanza.

Le classi con suffisso *DOP*, *DO* e *DOF* estendono rispettivamente le classi corrispondenti *DataObjectProxy*, *DataObject* e *DataObjectFactory* presenti in BROADgen®: la prima contiene una serie di metodi con cui operare sul singolo record, la seconda è una classe che sfrutta il pattern "Memento" e che serve a memorizzare lo stato di ogni singolo oggetto, mentre l'ultima fa uso del pattern "Factory", il quale fornisce un'interfaccia per creare un oggetto ma lascia che le sottoclassi decidano quale oggetto istanziare. Mentre le 3 classi descritte in precedenza vengono generate dal framework BROADgen®, viene poi scritta una classe per ogni BusinessObject, chiamata col nome generico "*NomeTabella.java"* dove vengono esplicitate le regole di business.

1.2 - L'ambito di intervento e gli obiettivi del progetto

La parte delle applicazioni Broadway Solutions Srl che necessita di essere rinnovata è quella relativa al modulo View: verrà cioè cambiata la modalità di renderizzazione delle varie viste.

Nella situazione di partenza il modulo View è costituito da:

- Componenti statiche
- Componenti dinamiche

Le componenti statiche sono le pagine *.jsp* (JavaServer Pages), le quali vengono mandate in esecuzione da un servlet container (Apache Tomcat in questo caso) e servono a costruire dinamicamente l'HTML delle pagine dal lato server.

Le componenti dinamiche invece sono costituite dai file dichiarativi *.xml* (eXtensible Markup Language) i quali definiscono l'interfaccia grafica, oggetto per oggetto, che dev'essere costruita al volo in fase di esecuzione.

Le viste utilizzate sono di 3 diversi tipi: QueryView, ListView e DetailView. La QueryView è la maschera di interrogazione in cui inserire i parametri di ricerca e contiene una serie di "ViewItem", che sono dei generici oggetti da visualizzare, essi possono essere semplici campi di testo od etichette come anche menu a tendina o selettori di data e ora. La ListView è, generalmente, la vista contenente il risultato di una query e contiene una lista di record mentre la DetailView è la vista che mostra il dettaglio di un singolo record.

Per quanto riguarda il ciclo di richiesta/risposta tra client e server le applicazioni Broadway Solutions Srl seguono il tradizionale pattern del web 1.0:

- Invio richiesta da parte del client
- Elaborazione della richiesta del server
- Invio risposta da parte del server
- Visualizzazione del risultato da parte del client

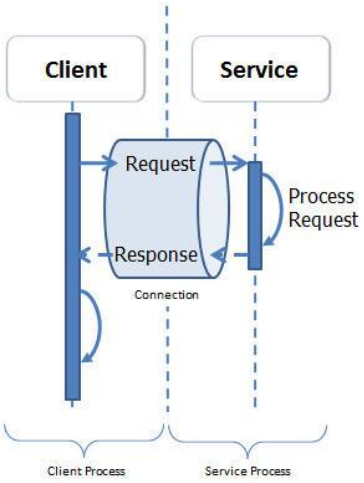

Figura 1.2 - Sequenza request/response nel web 1.0

Secondo questo meccanismo di request/response per ogni tipo di richiesta effettuata dal client, il server, dopo averla elaborata, ritrasmette l'intero codice HTML al client il quale per visualizzarla effettua il reload dell'intera pagina web.

Il modulo View verrà quindi rinnovato introducendo le caratteristiche tipiche del web 2.0, detto anche "web dinamico", in modo da far assomigliare sempre di più le web application alle applicazioni sviluppate in ambito desktop. Tra gli aspetti più significativi del web 2.0 consideriamo l'utilizzo di:

- AJAX
- Componenti più dinamici e interattivi all'interno dell'interfaccia grafica

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra il web browser e il server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra vengono richiesti al server e caricati in background attraverso l'uso delle "XMLHttpRequest" senza interferire con il comportamento della pagina esistente e sgravando il server dell'invio di risposte gravose in termini di tempo e dimensioni. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript.

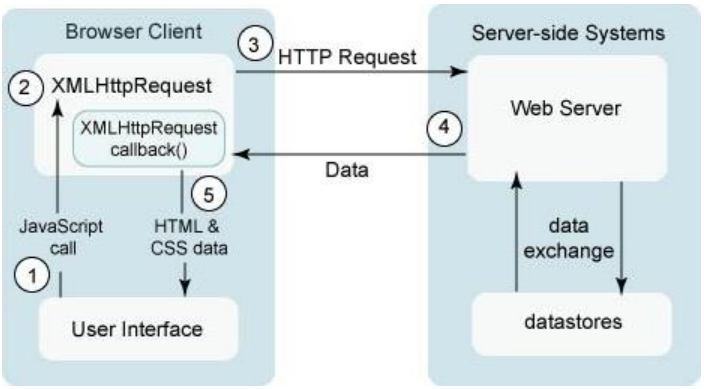

Figura 1.3 - **Sequenza request/response con AJAX**

Gli obiettivi richiesti alla progettazione del modulo BROADGWT riguardano fondamentalmente:

-L'aumento dell'efficienza delle richieste al server attraverso l'utilizzo preponderante di AJAX

-Il miglioramento, dal punto di vista estetico ed ergonomico, della interfaccia e della sua fruibilità

Uno dei motivi per cui si è richiesto l'intervento di rinnovamento dell'interfaccia grafica è dato dal fatto che i continui reload sulle pagine web comportano una scarsa ergonomia per l'utente finale, che assiste a continui "sfarfallii" sul proprio monitor e arriva ad utilizzare un'applicazione anche un'intera giornata lavorativa. AJAX ovvia a

questo problema poiché vengono ricaricate selettivamente soltanto alcune porzioni dell'interfaccia grafica.

Per quanto riguarda i componenti dell'interfaccia grafica, nello stato precedente la realizzazione del progetto le applicazioni Broadway Solutions Srl disponevano già di alcuni componenti dinamici, quali ad esempio le "longlist", campi di ricerca in cui mano a mano che si inseriscono caratteri vengono visualizzati una serie di risultati pertinenti in formato di una lista integrata, e le "foglie" del menu, cioè man mano che viene esplorato il menu dell'applicazioni vengono effettuate richieste asincrone per caricarlo. In aggiunta a questi sono stati inseriti componenti dinamici durante il caricamento delle singole porzioni di interfaccia (loader) e sono state estese le funzionalità dinamiche della longlist anche a componenti quali la shortlist ed altri. Infine sono state adottate tecniche esteticamente più moderne ed accattivanti quali effetti di transizione, ombreggiature etc grazie a una modifica sostanziale dei fogli di stile CSS.

Capitolo 2 - Tecnologie utilizzate

In questo capitolo verrà introdotta una panoramica sulle varie tecnologie incontrate ed utilizzate durante lo svolgimento del tirocinio.

- **Apache Tomcat 6**

Figura 2.1 – **Logo Apache Tomcat**

Apache Tomcat (o semplicemente Tomcat) è un contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma per l'esecuzione di applicazioni web sviluppate nel linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache. In passato, Tomcat era gestito nel contesto del Jakarta Project, ed era pertanto identificato con il nome di Jakarta Tomcat; attualmente è oggetto di un progetto indipendente. Tomcat è rilasciato sotto la Licenza Apache, ed è scritto interamente in Java; può quindi essere eseguito su qualsiasi architettura su cui sia installata una JVM (Java Virtual Machine). [3]

Nell'ambito Broadway Solutions Srl, le web application sviluppate costituiscono tecnicamente dei "context" Tomcat.

Sito Ufficiale: http://tomcat.apache.org/

Documentazione: http://tomcat.apache.org/tomcat-6.0-doc/index.html

Librerie maggiormente utilizzate: Ant, Commons Beanutils (v 1.7), Commons Collections (v 2.1), Commons Digester (v 1.7), Commons Fileupload (v 1.0), Http Components (v. 4.1.2), POI.

- **PostgreSQL 9**

Figura 2.2 – **Logo PostgreSQL**

PostgreSQL è un sistema per la gestione di basi di dati relazionali (RDBMS) ad oggetti rilasciato con licenza libera. Spesso viene abbreviato come "Postgres", sebbene questo sia un nome vecchio dello stesso progetto. E' di sicuro uno tra i più

avanzati database relazionali open-source con caratteristiche enterprise. PostgreSQL è una reale alternativa sia rispetto ad altri prodotti liberi come MySQL, Firebird SQL e MaxDB che a quelli a codice chiuso come Oracle, Informix o DB2 ed offre caratteristiche uniche nel suo genere che lo pongono per alcuni aspetti all'avanguardia nel settore dei database.

Nell'ambito di Broadway Solutions Srl PostgreSQL è usato per creare, manipolare e gestire i database che contengono i dati dei vari clienti o aziende. Con esso si interfacciano le librerie JDBC a livello applicativo.

Sito Ufficiale: http://www.postgresql.org/

Documentazione: http://www.postgresql.org/docs/9.0/static/index.html

- **Java SE e EE 7**

Figura 2.3 – **Logo Java**

Java è un linguaggio di programmazione orientato agli oggetti (OOP), fortemente tipizzato, indipendente dalla piattaforma in cui viene utilizzato e progettato per eseguire codice da sorgenti remote in modo sicuro. E' di fatto il linguaggio di elezione per lo sviluppo di applicativi gestionali server-side, soprattutto in ambito non Windows-based, nonché uno dei linguaggi più diffusi. La distribuzione Enterprise aggiunge al già vasto repertorio di librerie un'altra serie di funzionalità che estendono le potenzialità del linguaggio di programmazione.

In questo progetto Java è stato utilizzato con tutte le sue ultime caratteristiche più avanzate, tra le quali i generics e le annotations, per l'implementazione di tutti i componenti del modulo BROADGWT di interfaccia grafica, sfruttando le librerie e gli strumenti di Google Web Toolkit.

Sito Ufficiale: http://www.oracle.com/technetwork/java/index.html

Documentazione: http://docs.oracle.com/javase/7/docs/index.html

Librerie maggiormente utilizzate: Librerie base, Java I/O, JDBC, Java Networking.

- **Design Patterns**

Un design pattern (schema di progettazione) può essere definito "una soluzione progettuale generale a un problema ricorrente". Esso non è una libreria o un componente di software riusabile, quanto piuttosto una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software. I design pattern orientati agli oggetti tipicamente mostrano relazioni ed interazioni tra classi o oggetti, senza specificare le classi applicative finali coinvolte. Tali pattern risiedono quindi nel dominio dei moduli e delle interconnessioni. Ad un livello più alto sono invece i Pattern architetturali che hanno un ambito ben più ampio, descrivendo un pattern complessivo adottato dall'intero sistema.

Nello sviluppo delle proprie applicazioni la Broadway Solutions Srl fa largo uso di alcuni tra questi patterns, i quali possono essere suddivisi fondamentalmente in 3 categorie:

- Pattern creazionali
- Pattern strutturali
- Pattern comportamentali

La prima categoria è costituita da pattern che si occupano di nascondere i costruttori delle classi e sostituirli creando un'interfaccia. In questo modo si possono utilizzare oggetti nascondendo i dettagli implementativi sulla costruzione dell'oggetto e lasciando decidere a posteriori quale tipo di oggetto creare, come già visto nel Capitolo 1. Tra questi ad esempio l'"Abstract Factory", il "Factory Method", la "Lazy Inizialization" e il "Singleton". Per quanto riguarda i pattern strutturali, che consentono di riutilizzare oggetti esistenti, si può menzionare l'"Adapter", il "Composite", il "Decorator", il "Facade" e il "Proxy". Con riferimento ai pattern comportamentali, cioè quelli che risolvono problematiche relative alle interazioni più usuali tra gli oggetti, vengono utilizzati, per esempio, il "Command", l'"Observer" e il "Visitor". Oltre a questi ritroviamo i già citati pattern architetturali MVC e la Reflection.

- **HTML 4.01**

L'HTML (HyperText Markup Language) è il linguaggio solitamente usato per i documenti ipertestuali disponibili nel World Wide Web. In tali documenti, un tratto di testo può essere contrassegnato inserendo delle etichette, tag, che ne possono descrivere tra le altre cose la funzione, il colore, le dimensioni, il link o altre

caratteristiche. Il contenuto servito dai siti web in seguito a una richiesta dell'utente solitamente consiste di un documento HTML e dei file ad esso correlati che un web browser scarica da uno o più web server per elaborarli, interpretando il codice, al fine di generare la visualizzazione della pagina desiderata sullo schermo del computer. L'HTML non è un linguaggio di programmazione (in quanto non prevede alcuna definizione di variabili, strutture dati, funzioni, strutture di controllo) ma solamente un linguaggio di markup che descrive le modalità di impaginazione, formattazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web attraverso tag di formattazione. Tuttavia, l'HTML supporta l'inserimento di script e oggetti esterni quali immagini o filmati. Punto HTML (.html) o punto HTM (.htm) è anche l'estensione comune per riconoscere i documenti in questo formato.

Nelle applicazioni Broadway Solutions Srl basate sul web 1.0 esso veniva costruito blocco per blocco dal metodo *render()*, mentre con il progetto in esame il codice HTML verrà "scritto" direttamente dalle classi di GWT.

Sito Ufficiale: http://www.w3.org/html/

Specifiche: http://www.w3.org/TR/html4/

- **XML 1.0**

XML (eXtensible Markup Language) è un linguaggio di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. Costituisce il tentativo di produrre una versione semplificata di Standard Generalized Markup Language (SGML) che consenta di definire in modo semplice nuovi linguaggi di markup da usare in ambito web. Il nome indica quindi che si tratta di un linguaggio marcatore (markup language) estensibile (eXtensible) in quanto permette di creare tag personalizzati. Rispetto all'HTML, l'XML ha uno scopo ben diverso: mentre il primo definisce una grammatica per la descrizione e la formattazione di pagine web e, in generale, di ipertesti, il secondo è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati. Mentre l'HTML ha un insieme ben definito e ristretto di tag, con l'XML è invece possibile definirne di propri a seconda delle esigenze.

Viene utilizzato in Broadway Solutions Srl per i principali file di configurazione applicativa e per la dichiarazione delle interfacce grafiche.

Sito Ufficiale: http://www.w3.org/XML/

Specifiche: http://www.w3.org/TR/2008/REC-xml-20081126/

- **JSON**

Figura 2.5 – **Logo JSON**

E' la notazione di riferimento per la rappresentazione di gerarchie di oggetti "javascript-like" in forma testuale e per la trasmissione di dati strutturati generici su Web. La semplicità di JSON (JavaScript Object Notation) ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX. Il suo uso tramite JavaScript è particolarmente semplice, infatti l'interprete è in grado di eseguirne il parsing tramite una semplice chiamata alla funzione *eval()*. Questo fatto lo ha reso velocemente molto popolare a causa della diffusione della programmazione in JavaScript nel mondo del Web. JSON è costituito prettamente da 2 tipi di strutture:

- Un oggetto: un'insieme di coppie chiave/valore, realizzata ad esempio con una hashmap o un dizionario
- Una lista ordinata di valori: tipicamente realizzata tramite array o lista.

E' stato adottato da Broadway Solutions Srl per lo scambio di dati tra il modulo BROADGWT e l'Application server.

Sito Ufficiale: http://json.org/

Librerie utilizzate da Java: http://www.json.org/java/index.html

http://jackson.codehaus.org/

- **CSS 2 e 3**

Il CSS (Cascading Style Sheets o Fogli di stile) è un linguaggio informatico usato per definire la formattazione di documenti HTML, XHTML e XML ad esempio in siti web e relative pagine web. Le regole per comporre il CSS sono contenute in un insieme di

direttive (Recommendations) emanate a partire dal 1996 dal W3C. L'introduzione del CSS si è resa necessaria per separare i contenuti dalla formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine HTML che per gli utenti. Il CSS, inoltre, possiede una caratteristica tipica dei linguaggi object-oriented qual è l'ereditarietà: si possono infatti definire classi e selettori in cui specificare delle proprietà, con relativo valore, tramite i tag propri del CSS. Le classi o selettori assegnati ad un certo elemento del DOM (Document Object Model) verranno di fatto ereditate anche da tutti i nodi figli.

Nel progetto di BroadGWT lo styling CSS è una delle parti fondanti dal punto di vista delle possibilità di personalizzazione delle varie applicazioni. Pertanto è stato completamente ristrutturato rispetto ai fogli di stile utilizzati nelle vecchie web apps , e, come verrà spiegato nel capitolo seguente, è stata creata una struttura in cui vengono associati fogli di stile specifici ad una data applicazione.

Sito Ufficiale: http://www.w3.org/TR/1998/REC-CSS2-19980512/Overview.html

- **ECMAscript (JavaScript)**

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Oracle. JavaScript è stato standardizzato per la prima volta tra il 1997 e il 1999 dalla ECMA con il nome ECMAScript. A differenza di Java, JavaScript non viene compilato ma interpretato all'interno di un web browser che è dotato di un proprio motore per l'esecuzione degli script (JavaScript engine). Inoltre è un linguaggio scarsamente tipizzato (loosely typed) e senza un vero e proprio concetto di ereditarietà. Il JavaScript generalmente agisce sul DOM della pagina web oppure altera proprietà CSS.

Nel progettare un modulo web 2.0 JavaScript è uno degli strumenti fondamentali in quanto il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il server non viene sovraccaricato a causa delle richieste dei client, migliorando notevolmente l'efficienza delle applicazioni. Tuttavia la programmazione ma soprattutto il debugging di codice JavaScript comportano più di qualche difficoltà, ed è per questo che si è scelto di utilizzare lo strumento proposto da Google™, Google Web Toolkit, che verrà esaminato nel seguito.

Sito Ufficiale: http://www.ecmainternational.org/publications/standards/Ecma-262.htm

Specifiche: http://www.ecmainternational.org/publications/files/ECMA-ST/Ecma-262.pdf

- **GWT 2.4**

Figura 2.6 – **Logo GWT**

Google Web Toolkit (GWT) è un toolkit di sviluppo per la costruzione e l'ottimizzazione di applicazioni complesse basate su browser web. GWT è utilizzato in molti prodotti di Google, tra cui Google AdWords. Esso permette agli sviluppatori di programmare e utilizzare codice sorgente Java e ottenere del codice compilato in JavaScript anziché in bytecode. E' comunque possibile utilizzare le funzionalità dirette di JavaScript nel codice sorgente utilizzando l'interfaccia nativa JSNI (JavaScript Native Interface). Il codice sorgente prodotto può essere facilmente sottoposto al debug senza essere compilato grazie alla modalità di esecuzione di sviluppo (Development Mode) che consente inoltre la sostituzione di codice "a caldo" (hot swapping), cioè senza riavviare l'esecuzione dell'applicazione ma con una semplice operazione di refresh. Al momento della compilazione invece viene creata una serie di file javascript (permutations), ognuno con degli accorgimenti specifici per essere eseguito su ciascuno dei maggiori browser in circolazione, in modo da ovviare ai noti problemi di compatibilità cross-platform.

Nel progetto di BROADGWT, GWT è stato utilizzato per costruire la nuova interfaccia grafica che utilizza le strutture ed i controlli della propria libreria di widget. Inoltre si sono sfruttate le librerie per la parsificazione dei dati provenienti dal server sotto forma di oggetti JSON.

Sito Ufficiale: http://developers.google.com/web-toolkit/

Guida per gli sviluppatori: https://developers.google.com/web-toolkit/doc/2.4/DevGuide?hl=it-IT

API: http://google-webtoolkit.googlecode.com/svn/javadoc/latest/index.html?overview-summary.html

- **Guice e GIN**

Guice è il framework open source per la “dependency injection” (iniezione delle dipendenze) distribuito da Google e compatibile con la versione 1.5 di Java e seguenti, mentre GIN (Gwt INjection) è la versione lato client costruita appositamente per GWT. La dependency injection è un un pattern della programmazione ad oggetti di tipo creazionale, che possiamo pensare come un’ evoluzione del pattern abstract factory. Infatti dove prima l’abstract factory riusciva a disaccoppiare il processo di utilizzo di un oggetto da quello della sua creazione e di conseguenza implementazione, la dependency injection riesce nello stesso intento utlizzando un’unica classe “Container”, che, in base a come viene configurata e grazie all’uso delle annotazioni “Inject”, evita la necessità della costruzione delle abstract factory, associando direttamente la classe implementativa corretta.

Nell’implementazione del modulo 2.0 viene utilizzata l’annotation “Inject” di GIN per segnalare al compilatore quali elementi (classi, campi o metodi) devono essere “iniettati” a runtime da GIN stesso.

Sito Ufficiale Guice: http://code.google.com/p/google-guice/

Sito Ufficiale GIN: http://code.google.com/p/google-gin/

- **Eclipse Indigo 3.7**

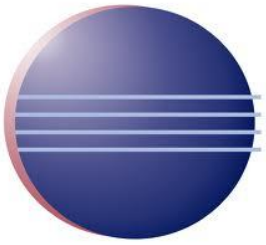

Figura 2.7 – **Logo Eclipse**

Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation sullo stile dell'open source. Eclipse può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, Javascript, PHP e altri. La piattaforma di sviluppo è incentrata sull'uso di plug-in, delle componenti software ideate per uno specifico

scopo, per esempio la generazione di diagrammi UML, ed in effetti tutta la piattaforma è un insieme di plug-in, versione base compresa, e chiunque può sviluppare e modificare i vari plug-in. Nella versione base è possibile programmare in Java, usufruendo di comode funzioni di aiuto quali: completamento automatico ("Code completion"), suggerimento dei tipi di parametri dei metodi, possibilità di accesso diretto a CVS e riscrittura automatica del codice (funzionalità questa detta di Refactoring) in caso di cambiamenti nelle classi. Essendo scritto in Java, Eclipse è disponibile per le piattaforme Linux, HP-UX, AIX, Mac OS X e Windows. [4]

E' il principale strumento di sviluppo utilizzato per il progetto. Gli applicativi Broadway Solutions Srl utilizzano dei plugin custom specifici che ne estendono le funzionalità, e grazie all'integrazione con il plugin di GWT per Eclipse è possibile utilizzare le funzionalità di debugging e deployment delle applicazioni create con GWT direttamente in Eclipse. E' inoltre interessante notare l'utilizzo del repository CVS per tenere sincronizzato il lavoro svolto.

Sito Ufficiale: http://www.eclipse.org

Documentazione: http://help.eclipse.org/indigo/index.jsp

Capitolo 3 – Lavoro svolto

3.1 - Prima parte

La prima parte del progetto, seguita al periodo di formazione, è stata svolta dal tirocinante in stretta collaborazione con il tutor ed il personale tecnico dell'azienda.

3.1.1 - Strutturazione del modulo BROADGWT

In questa fase si è costruita la struttura del nuovo modulo 2.0 e si è scelto un'applicazione già esistente, prodotta da Broadway Solutions Srl, su cui testare la nuova interfaccia grafica. L'applicazione in questione si tratta di "FoodLab", un gestionale creato per aziende che si occupano di certificazioni sanitarie nel settore alimentare. Si è quindi creata una versione "2.0" di FoodLab, denominata "FoodLabGWT", la quale si appoggia al modulo BROADGWT per costruire la propria struttura ma viene poi personalizzata dalle proprie versioni di alcuni file implementativi e dei file .css oltreché essere costruita, a livello d'interfaccia, dai propri file .xml dichiarativi.

Un'applicazione standard creata con GWT è strutturata nel modo seguente:

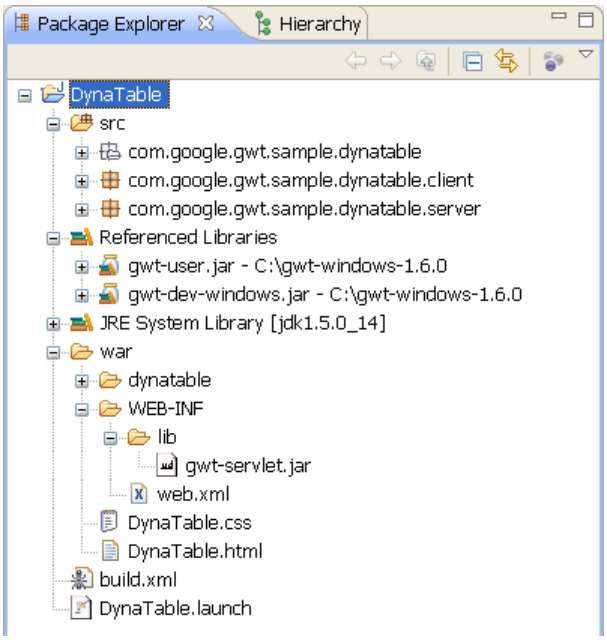

Figura 3.1 – Struttura package in un'applicazione GWT

Come si può notare, oltre ai package (cartelle) con i file delle librerie, le due cartelle fondamentali sono:

- src
- war

La prima contiene il codice sorgente dell'applicazione, la seconda è la cartella contenente le risorse statiche e i file compilati, nonché la cartella che verrà poi utilizzata per il "deployment" dell'applicazione stessa. Nella cartella "src" è stata fatta un'ulteriore suddivisione del codice sorgente creando le cartelle client (che dovranno contenere il codice eseguito sul lato client), server (codice che verrà eseguito lato server) e shared (codice utilizzato da entrambi i componenti). Nella cartella client, inoltre, sono stati creati altri packages, prendiamo ad esempio "api" o "gwt", per distinguere il codice appartenente a quella categoria: in questo caso file di interfaccia e implementazioni delle stesse.

3.1.2 - Integrazione con JSON

Nel nuovo modulo, come già anticipato, è stato introdotto l'utilizzo di JSON come strumento di scambio di dati tra server e client alternativo all'XML, usato precedentemente nelle vecchie applicazioni. Per fare ciò si è definita un'interfaccia, chiamata "JSONParsable", implementata dalle classi utilizzate lato server, che attraverso un metodo, "*fillJson()*", costruiscono i cosiddetti "JSONObject" sotto forma di array di coppie chiave/valore, inserendovi le proprietà degli oggetti da visualizzare (ViewItem) da passare al client. Nelle classi lato client, d'altro canto, i campi che sarebbero stati "parsificati" dal JSON in formato testuale sono stati contraddistinti con una annotation creata appositamente, chiamata "Parsed". Infine sono state create alcune classi lato client, simili come funzionalità alle loro corrispettive classi nelle applicazioni vecchio tipo, per interfacciarsi con i nuovi oggetti JSON: tra queste menzioniamo ad esempio "JSONWidget", corrispettiva di "UIControl", che raccoglie le proprietà del ViewItem e invoca il metodo per la visualizzazione di un widget specifico tramite il metodo "*getWidget()*"; a seconda che un widget sia editabile o meno ne fornisce la versione in sola lettura con "*getWidgetReadOnly()"*. Qui di seguito un estratto dal codice di JSONWIdget con gli elementi descritti in precedenza:

…

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
```

```
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
public abstract class JSONWidget implements JSONParsable<JSONWidget>, JSONPostable {

        @Parsed String fId;
        @Parsed String fValue;
        @Parsed String fPrompt;
        @Parsed String fPostPrompt;
        @Parsed String fHint;
        @Parsed boolean fReload;
        @Parsed boolean fEditable;
        @Parsed boolean fMandatory;
        @Parsed(defaultBooleanValue=true) boolean fNewLine;
        @Parsed int fSize;
        @Parsed int fPromptSize;
        @Parsed int fPostPromptSize;
        @Parsed String fContentClass;
        Widget fPromptWidget;
…

        public      JSONWidget      init(JSONObject      pObject,      JSONViewStateFactory
pViewStateFactory){
                fViewStateFactory = pViewStateFactory;
                parse(pObject);
                return this;
        }

        public abstract Widget getBaseWidget();

        public Widget getWidget() {
                Widget mRet = null;
                if (!isEditable()) {
                        mRet = getWidgetReadOnly();
                } else {
                        mRet = getBaseWidget();
                }
                if(isReload()) {
                        handleReloadRequest(mRet);
                }
                if(isEditable() && isMandatory()){
                        HorizontalPanel mPanel = new HorizontalPanel();
                        mPanel.add(mRet);
                        mPanel.add(new Label("*"));
                        mRet = mPanel;
                }
                return mRet;
        }
public Widget getWidgetReadOnly() {
                Label mLabel = new Label();
                if (getValue() != null && getValue().length()>0)
                        mLabel.setText(getValue());
                return mLabel;
        }
        public Widget getPromptWidget() {
                if (fPromptWidget == null) {
                        String mPrompt = getPrompt();
                        if (mPrompt == null || mPrompt.length()==0)
                                return null;
```

```
                        fPromptWidget = new Label(mPrompt);
                        fPromptWidget.setStyleName("prompt");
                        fPromptWidget.setWidth("200px"); //promptSize
                }
                return fPromptWidget;
        }
```

…

Un’altra caratteristica interessante implementata durante la prima fase è un “servizio di messaggistica” tra i vari componenti dell’applicazione: ciascun componente può essere o meno in ascolto di nuovi messaggi e, a seconda che sia interessato al messaggio in circolazione, invia come riposta “INTERESTED” oppure “NOT INTERESTED”. Nel caso che tale componente sia interessato, il messaggio viene classificato come “USED”. Qui una porzione del codice del “UIMessageBroker”, classe dichiarata come Singleton in modo che ogni componente faccia sempre riferimento alla medesima istanza:

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
@Singleton
public final class UIMessageBroker {

        List<UIMessageListener> fListeners = new ArrayList<UIMessageListener>();

        public void addUIListener(UIMessageListener pListener) {
                fListeners.add(pListener);
        }

        public void removeUIListener(UIMessageListener pListener) {
                fListeners.remove(pListener);
        }

        public void sendUIMessage(UIMessage pMessage) {
                for (UIMessageListener mListener : new
ArrayList<UIMessageListener>(fListeners)) {
                        try {
                                UIMessageResponse mResponse =
mListener.notifyUIMessage(pMessage);
                                switch (mResponse) {
                                        case USED:
                                                break;
                                        case FAILED:
                                                break;
                                        case CONSUMED:
                                                // if adding logic to USED, add here, too
                                                return;
                                        case NOT_INTERESTED:
                                                break;
                                        default:
```

```
                                        break;
                                }
                        } catch (Throwable t) {
                                JSONObject mErrorObj = new JSONObject();
                                mErrorObj.put(MESSAGE_TYPE, new
JSONString(INTERNAL_CLIENT_ERROR));
                                mErrorObj.put(MESSAGE, new JSONString(t.toString()));
                                UIMessage mMessage = GWT.create(UIMessage.class);
                                sendUIMessage(mMessage.setJSONMessage(mErrorObj));
                        }
                }
        }

}
```

Per quanto riguarda invece il meccanismo con cui vengono istanziati i widget creati con GWT al posto di quelli dell'interfaccia grafica precedente, è stata sviluppata un'annotation che fa da ponte tra i vecchi ed i nuovi "controlli". E' stata denominata "UIControlMapping" e viene posta subito prima la dichiarazione della classe, nella fattispecie un JSONWidget, e deve contenere come parametro il nome esatto del controllo grafico appartenente alla vecchia interfaccia grafica. Vediamo a titolo d'esempio tale funzionamento con la classe DateJSONWidget:

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
@UiControlMapping("Date")
public abstract class DateJSONWidget extends JSONWidget
```

Il widget viene poi istanziato a runtime, restituito dal metodo *getBaseWidget()* che sovrascrive quello della superclasse JSONWidget.

```
@Override
public Widget getBaseWidget() {
        // Create a DateBox
…

return mDateBox;
}
```

Al termine della prima fase della progettazione si è implementato anche il sistema di late binding e personalizzazione dell'interfaccia in base all'applicazione specifica, per esempio lo stesso FoodLab. Il late binding è una peculiarità di GWT che ha un funzionamento simile al pattern Abstract Factory a parte il fatto di essere eseguita in fase di compilazione anziché di esecuzione. Essa permette al compilatore di associare a un'interfaccia un'implementazione specifica in Java al momento dell'invocazione del metodo *GWT.create()* e quindi permette di risparmiare risorse in termini di spazio con il fatto che verrà compilata in JavaScript solo l'implementazione d'interesse.

3.2 - Seconda parte

Nella seconda parte dello stage l'implementazione del modulo BROADGWT è proseguita autonomamente da parte del tirocinante, seguendo le indicazioni del tutor aziendale.

3.2.1 - Costruzione dell'interfaccia grafica

La costruzione dell'interfaccia grafica è stata eseguita per mezzo delle classi di widget e pannelli messi a disposizione da GWT. Per quanto riguarda i "panels", essi sono dei contenitori di widget e altri pannelli: è stato fatto uso dell'evoluzione di tali classi, chiamate "LayoutPanels", implementate a partire dalla versione 2.0 di GWT. Inizialmente si è costruito il layout generale della pagina web attraverso la classe "StandardLayoutFactory" che sfrutta, attraverso il metodo *getLayout(),* il "DockLayoutPanel" per comporre le varie parti della pagina web: è stata infatti aggiunta una parte superiore con il titolo dell'applicazione, una inferiore e uno "splitter" che divida il menu dal contenuto vero e proprio, con la possibilità di ridimensionamento in caso non si voglia visualizzare il menu. Qui un estratto di codice di tale classe con l'implementazione del metodo getLayout():

```
	/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
```

```
* distribute this software for any purpose
* without explicit permission granted by
* BROADway Solutions srl.
*/
	@Override
	public Widget getLayout() {
		final DockLayoutPanel mPanel = new DockLayoutPanel(Unit.PX);
		mPanel.setHeight(Window.getClientHeight()+"px");
		mPanel.setWidth(Window.getClientWidth()+"px");

		mPanel.addNorth((Widget)fTopPanel.getView(), 50);
		mPanel.addSouth((Widget)fBottomPanel.getView(), 20);
		SplitLayoutPanel mMiddle = new SplitLayoutPanel(6);
		//		VerticalPanel mVP = new VerticalPanel();
		//		mVP.add((Widget)fContentPanel.getView());

		mMiddle.addWest(new ScrollPanel((Widget)fMenuPanel.getView()), 210);
		mMiddle.add(new ScrollPanel((Widget) fContentPanel.getView()));
		mPanel.add(mMiddle);


		Window.addResizeHandler(new ResizeHandler() {
			@Override
			public void onResize(ResizeEvent pEvent) {
				mPanel.setHeight(pEvent.getHeight()+"px");
				mPanel.setWidth(pEvent.getWidth()+"px");
			}
		});
		return mPanel;
	}
```

Qui mettiamo a confronto il layout delle vecchie applicazioni con quello di BROADGWT con la schermata iniziale di login:

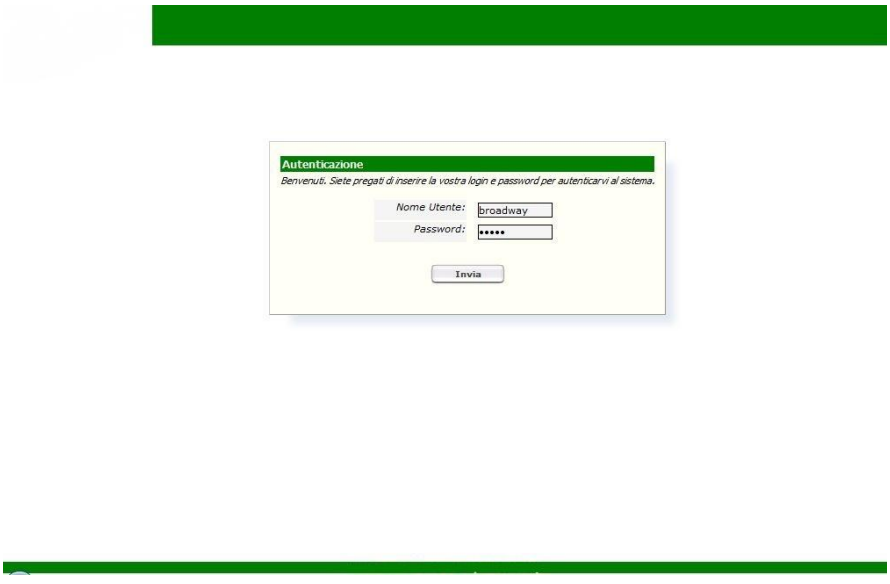

Figura 3.2 – **Pagina di login 1.0**

Figura 3.3 – **Pagina di login con BROADGWT**

In seguito si è passati all'implementazione del menu, dimensionato staticamente affinché occupi 200 pixel sulla parte sinistra della pagina, utilizzando la classe "StackLayoutPanel" di GWT; tale classe permette di porre una serie di elementi in pila verticalmente visualizzandone uno alla volta, ognuno con un header su cui l'utente può cliccare per mostrarlo. E' stato creato quindi l'albero degli elementi del menu inserendo iterativamente tutti gli elementi dai nodi padre fino alle foglie più esterne:

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
public class MenuPanelView extends DecoratedStackPanel implements IMenuView {

	@Inject private JSONRequestBuilder fRequestBuilder;
	//	private final BroadCssResource fCss = BroadGWT.CSS.BroadGWT();

	@Override
	public void clearContents() {
		clear();
	}

	@Override
	public void init() {
		setWidth("200px");
```

```
                addStyleName(BroadGWT.CSS.BroadGWT().menuPanel());
        }

        @Override
        public void addTree(JSONTreeItem pRootTree) {
                for (JSONTreeItem mElem : pRootTree.fChildren) {
                        Tree tree = new Tree();
                        JSONTreeItem[] mChildrenElem = mElem.fChildren;
                        if(mChildrenElem!=null){
                                for (JSONTreeItem mChildElem : mChildrenElem) {
                                        TreeItem                mChild                  =
tree.addItem(writeItem(mChildElem));
                                        writeChildren(mChildElem, mChild);
                                }
                        }
                        add(tree, mElem.fDescription, true);
                }

        }
```

Di seguito si può notare la differenza tra le due interfacce del menu, quella delle vecchie applicazioni, nella quale comunque venivano effettuate richieste AJAX per aprire il contenuto delle foglie più esterne, e la versione "2.0":

Figura 3.4 – **Vecchio menu**

Figura 3.5 – **Menu 2.0**

Nel lavoro di implementazione dell'interfaccia si è poi passati alla costruzione dei tre tipi di vista fondamentali: query, lista e dettaglio, in cui si è mantenuta, esteticamente parlando, una certa attinenza con quanto era stato fatto nella precedente interfaccia grafica.

Dopo aver costruito le parti fondamentali dell'interfaccia si è passati allo sviluppo dei vari Widget di BROADGWT che sarebbero poi andati a sostituire gli "UIControl" già presenti in BROADgen®. E' stato fatto una sorta di "censimento" di tutti gli UiControl utilizzati nelle varie applicazioni e, per ognuno, è stata usata l'annotazione "@Uses" per contrassegnare quali proprietà del ViewItem (oggetto da visualizzare) fossero effettivamente utilizzate da uno specifico controllo. In questo modo i valori di tali proprietà vengono passati dai vecchi controlli ai nuovi widget tramite il metodo *fillJson()* e i JSONWidget leggono tali valori utilizzando l'annotazione "@Parsed"; per quanto concerne la "mappatura" tra due controlli corrispondenti essa avviene con l'annotazione "@UIControlMapping" già vista in precedenza.

Di seguito riportiamo un esempio di implementazione del semplice controllo "HTMLJSONWidget": questo controllo serve a visualizzare nella pagina una porzione di codice HTML passato in forma testuale e viene utilizzata una "TextBox" (casella di testo) per visualizzarne il contenuto:

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
package it.broadwaysolutions.gwt.client.jsonwidget;

import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.Widget;

@UiControlMapping({"Html","HtmlFormat"})
public abstract class HtmlJSONWidget extends JSONWidget {

        @Override
        public Widget getWidget() {
                final HTML mTextBox = new HTML();
                if (getValue() != null && getValue().length()>0)
                        mTextBox.setHTML(getValue());
                return mTextBox;
        }
```

3.2.2 - Styling CSS e meccanismo di personalizzazione

Dopo aver costruito l'interfaccia e implementato i vari widget si è passati all'aspetto più prettamente estetico della progettazione di BROADGWT: lo styling CSS.

La realizzazione dei fogli di stile è stata ottenuta grazie ad un'apprezzabile funzionalità di GWT: la classe "CSSResource". Tramite questa risorsa è possibile definire in una classe java che estenda CSSResource i nomi delle classi CSS che vengono assegnati all'interno del codice Java nelle varie porzioni dell'interfaccia grafica e che verranno poi utilizzate nel foglio di stile vero e proprio; queste classi vengono "minificate" cioè compresse e ottimizzate per ottenere il minor peso possibile in termini di dimensione del file css. Nel caso di BROADGWT è stato creato il file BroadCSSResource:

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
package it.broadwaysolutions.gwt.client;

import com.google.gwt.resources.client.CssResource;

public interface BroadCssResource extends CssResource {

        String queryButtonBar();
        String breadcrumb();
        String contentPanel();
        String masterLayoutTable();
        String master_header();

        String listTable();
        String listPromptLabel();
        String oddRow();
        String evenRow();

        String dialogContent();

        String loginBoxRow();
        String loginBoxRowLabel();
        String loginSubtitle();

        String prompt();

        String topPanel();
        String logo();
        String topLabel();
        String bottom();
```

```
        String menuPanel();

}
```

Per ciò che concerne la personalizzazione del CSS in base all'applicazione specifica , è stato implementato il file "AppStyle.css", che assegna un valore alle costanti definite in "StandardResources", un file che estende "ClientBundle". Tali costanti sono definite sia per BROADGWT che per qualsiasi altra applicazione che venga generata con esso, ma ogni applicazione può sovrascrivere i valori assegnati a tali costanti implementando un proprio AppStyle.css che comporta la personalizzazione di font, colori, gradienti etc.. a seconda delle esigenze specifiche dell'applicazione. Qui di seguito vediamo la versione base di questo file presente in BROADGWT, con la definizione di alcune costanti.

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
/*
This css file is meant to be overridden by applications
with their custom colors and act just for reference
for what variables can be customized
*/

@def MAIN_TEXT_COLOR black;
@def DEF_MARGIN 0.2em;
@def DEF_PADDING 0.15em;
@def DEF_BORDER_RADIUS 0.4em;
@def DEF_WIDG_BACKGROUND #f0f0f6;
@def DEF_WIDG_BACKGROUND_HOVER #fafafa;
@def DEF_HEADER_BACKGROUND #FFFFFF;
@def DEF_TAB_ITEM_BACKGROUND                                    #0A6B0A;
@def DEF_TAB_ITEM_BACKGROUND_SELECTED                   #0A6B0A;
@def DEF_TAB_ITEM_BACKGROUND_HOVER                                      #33C833;
@def DEF_TAB_ITEM_BACKGROUND_SELECTED_HOVER             #3d9e3d;
@def DEF_HEADER_BORDER_RADIUS literal("10px 10px 0 0");
@def DEF_MENU_WIDTH 200px;
@def  DEF_SPLITTER_BACKGROUND_WEBKIT_GRADIENT  literal("-webkit-linear-gradient(top,  white
20%, #008500 50%, white 80%)");
@def  DEF_SPLITTER_BACKGROUND_WEBKIT_GRADIENT_HOVER  literal("-webkit-linear-gradient(top,
white 20%, #54B454 50%, white 80%)");
@def  DEF_SPLITTER_BACKGROUND_WEBKIT_GRADIENT_ACTIVE  literal("-webkit-linear-gradient(top,
white 20%, #0A6B0A 50%, white 80%)");
@def  DEF_SPLITTER_BACKGROUND_MOZ_GRADIENT  literal("-moz-linear-gradient(top,  white  20%,
#008500 50%, white 80%)");
@def  DEF_SPLITTER_BACKGROUND_MOZ_GRADIENT_HOVER  literal("-moz-linear-gradient(top,  white
20%, #54B454 50%, white 80%)");
```

```
@def DEF_SPLITTER_BACKGROUND_MOZ_GRADIENT_ACTIVE literal("-moz-linear-gradient(top, white
20%, #0A6B0A 50%, white 80%)");
@def DEF_SPLITTER_BACKGROUND_IE_GRADIENT literal("-ms-linear-gradient(top, white 20%,
#008500 50%, white 80%)");
@def DEF_SPLITTER_BACKGROUND_IE_GRADIENT_HOVER literal("-ms-linear-gradient(top, white
20%, #54B454 50%, white 80%)");
@def DEF_SPLITTER_BACKGROUND_IE_GRADIENT_ACTIVE literal("-ms-linear-gradient(top, white
20%, #0A6B0A 50%, white 80%)");
@def LIST_BACKGROUND_HOVER #8080FF;
@def DEF_TOP_FONT literal("italic bold 16pt Georgia");
@def DEF_BOTTOM_FONT literal("bold 8pt Georgia");
@def DEF_MASTER_HEADER_FONT literal("bold 12px Tahoma");
@def DEF_SECTION_HEADER_FONT literal("bold 11px Tahoma");
@def DEF_HEADER_TEXT_COLOR white;
```

Per finire, viene presentato un estratto dal foglio di stile vero e proprio BroadGWT.css, dove è possibile notare l'uso delle annotazioni: "@eval " per valutare il valore di una certa costante definita in StandardResources, "@def" per definire una costante che abbia visibilità solo all'interno del foglio di stile e "@external" per indicare al compilatore che la classe utilizzata proviene da un file esterno e non da una CSSResource.

```
/*
 * Copyright (C) BROADway Solutions srl.
 * This software is property of BROADway Solutions srl.
 * You are not allowed to use, copy, modify and/or
 * distribute this software for any purpose
 * without explicit permission granted by
 * BROADway Solutions srl.
 */
@external .gwt-DecoratedStackPanel;
@external .gwt-DecoratedStackPanel .gwt-StackPanelContent;
@external .gwt-DecoratedStackPanel .gwt-StackPanelItem;
@external .gwt-DecoratedStackPanel .gwt-StackPanelItem-selected;
@external .errorMessage;
@external .confirmButton;
@external .infoNotificationBox;
@external .warning;
@external .section_header;
@external .sectionLabel;
@external .scrollRichTextArea;
@external .HTMLRichText;
@external .progressBar;
@external .progressBarContainer;
@external .progressBarBgText;
@external .progressBarText;

…
```

```
@eval                                                                    MAIN_TEXT_COLOR
it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().MAIN_TEXT_COLOR();
@eval DEF_MARGIN it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_MARGIN();
@eval DEF_PADDING it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_PADDING();
@eval                                                                 DEF_BORDER_RADIUS
it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_BORDER_RADIUS();
@eval                                                               DEF_WIDG_BACKGROUND
it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_WIDG_BACKGROUND();
@eval                                                         DEF_WIDG_BACKGROUND_HOVER
it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_WIDG_BACKGROUND_HOVER();
@eval                                                             DEF_HEADER_BACKGROUND
it.broadwaysolutions.gwt.client.BroadGWT.CSS.AppStyle().DEF_HEADER_BACKGROUND();

…


@def DEF_BORDER 1px solid #ccc;
@def DEF_BORDER_COLOR_HOVER #b5b5b5;
@def DEF_FONT bold 0.95em arial, sans-serif;
@def DEF_MAIN_TEXT_FONT  11px Verdana;
@def DEF_MENU_CONTENT_FONT  12px Verdana;
@def DEF_HEADER_HEIGHT 1em;
@def DEF_FOCUS_COLOR #E7E723;
@def DEF_LOGIN_BOX_SHADOW 10px 10px 10px #6D6D6D;
@def DEF_TABLE_BOX_SHADOW 5px 5px 5px #6D6D6D;

…

/* Minimize the GWT classic theme dialog box corner dimensions */

.gwt-DialogBox .dialogTopLeftInner,
.gwt-DialogBox .dialogTopRightInner,
.gwt-DialogBox .dialogBottomLeftInner,
.gwt-DialogBox .dialogBottomRightInner {
        width:0;
        height: 0;
}



.gwt-DialogBox {
        padding: 3px;
        background-color: DEF_WIDG_BACKGROUND;
        border: 1px solid;
        border-color:#ccc #999 #999 #ccc;

        border-radius: DEF_HEADER_BORDER_RADIUS;
        clip: inherit !important;
        -webkit-box-shadow: DEF_LOGIN_BOX_SHADOW;
        -moz-box-shadow: DEF_LOGIN_BOX_SHADOW;
        -ie-box-shadow: DEF_LOGIN_BOX_SHADOW;
        box-shadow: DEF_LOGIN_BOX_SHADOW;
}

.gwt-HTML {
        color: MAIN_TEXT_COLOR;

}
```

```
.progressBarContainer {
        background: DEF_WIDG_BACKGROUND;
        border: DEF_BORDER;
        height: 2em;
        position: relative;
        border-radius: DEF_PROGRESSBAR_BORDER_RADIUS;
        text-align: center;
}

.progressBarBgText {
        line-height: 2em;
        white-space: nowrap;
/*      width: 100%;
        border-radius: DEF_PROGRESSBAR_BORDER_RADIUS; */
}

.progressBar {
        background-color: DEF_TAB_ITEM_BACKGROUND;
        background-image: DEF_HEADER_BACKGROUND_WEBKIT_GRADIENT;
        background-image: DEF_HEADER_BACKGROUND_MOZ_GRADIENT;
        overflow: hidden;
        position: absolute;
        height: 100%;
        top: 0;
        border-radius: DEF_PROGRESSBAR_BORDER_RADIUS;
}

.progressBarText {
        position: absolute;
        top: 0;
        line-height: 2em;
        color: white;
}

…
```

Conclusioni

Il tirocinio ha portato alla realizzazione del modulo BROADGWT con buoni risultati. L'interfaccia risulta ergonomica, accattivante e facilmente fruibile da parte dell'utente finale, come negli obiettivi richiesti. Il modulo, testato sull'applicazione FoodLab, ha già implementate molte delle funzionalità presenti nella vecchia applicazione con un buon livello di efficienza nelle prestazioni.

Restano da risolvere alcune problematiche riguardo alla stabilità dell'applicazione che, soprattutto in Development Mode, lancia alcune eccezioni non previste. Inoltre, per quanto riguarda i widget, rimangono da implementare solamente alcuni controlli tra i più complessi, ad esempio quello per visualizzare una mappa di Google Maps o quello per i grafici a più dimensioni. L'aspetto più notevole di questo modulo resta comunque l'amplia possibilità di estenderlo e personalizzarlo a seconda delle necessità specifiche di ogni applicazione.

L'esperienza del tirocinio presso la Broadway Solutions Srl ha portato il tirocinante a prendere confidenza con strumenti e tecnologie a lui prima poco conosciuti permettendogli di incrementare il suo bagaglio tecnico e migliorare le sue capacità nella programmazione ad oggetti.

Bibliografia

[1] Il Pattern MVC, Francesco Camarlinghi - http://www.html.it/pag/18299/il-pattern-mvc/

[2] Model View Controller Pattern (MVC), Claudio De Sio Cesari - http://www.claudiodesio.com/ooa&d/mvc.htm

[3] Apache Tomcat - http://it.wikipedia.org/wiki/Apache_Tomcat

[4] Eclipse - http://it.wikipedia.org/wiki/Eclipse_%28informatica%29

Indice delle figure

Ringraziamenti

Per avermi fatto raggiungere questo importante traguardo il primo ringraziamento va alla mia famiglia, alla mia ragazza e ai miei amici che mi hanno supportato durante tutti gli anni del mio percorso e mi hanno spronato nell'ultimo periodo a concludere gli studi. Poi volevo ringraziare il mio relatore Sergio Congiu, che mi ha dato la sua disponibilità nell'assistermi durante lo svolgimento del tirocinio e della prova finale, e il mio tutor aziendale, Tiziano Callegari, che assieme alle altre affabili persone della Broadway Solutions Srl, hanno reso l'esperienza del mio tirocinio oltre che utile, interessante e proficua, anche piacevole e divertente.